Arithmetic

`add $d,$s,$t`	Add: $d = $s + $t
`addu $d,$s,$t`	Add unsigned: $d = $s + $t
`sub $d,$s,$t`	Subtract: $d = $s - $t
`subu $d,$s,$t`	Subtract unsigned: $d = $s - $t
`addi $t,$s,C`	Add immediate: $t = $s + C (signed)
`addiu $t,$s,C`	Add immediate unsigned: $t = $s + C (signed)
`mult $s,$t`	Multiply: LO = (($s * $t) << 32) >> 32; HI = ($s * $t) >> 32;
`multu $s,$t`	Multiply unsigned: LO = (($s * $t) << 32) >> 32; HI = ($s * $t) >> 32;
`div $s, $t`	Divide: LO = $s / $t HI = $s % $t
`divu $s, $t`	Divide unsigned: LO = $s / $t HI = $s % $t

Data Transfer

`lw $t,C($s)`	Load word: $t = Memory[$s + C]
`lh $t,C($s)`	Load halfword: $t = Memory[$s + C] (signed)
`lhu $t,C($s)`	Load halfword unsigned: $t = Memory[$s + C] (unsigned)
`lb $t,C($s)`	Load byte: $t = Memory[$s + C] (signed)
`lbu $t,C($s)`	Load byte unsigned: $t = Memory[$s + C] (unsigned
`sw $t,C($s)`	Store word: Memory[$s + C] = $t
`sh $t,C($s)`	Store half: Memory[$s + C] = $t
`sb $t,C($s)`	Store byte: Memory[$s + C] = $t
`lui $t,C`	Load upper immediate: $t = C << 16
`mfhi $d`	Move from high: $d = HI
`mflo $d`	Move from low: $d = LO
`mfcZ $t, $d`	Move from Control Register: $t = Coprocessor[Z].ControlRegister[$d]
`mtcZ $t, $d`	Move to Control Register: Coprocessor[Z].ControlRegister[$d] = $t

Logical

`and $d,$s,$t`	And: $d = $s & $t
`andi $t,$s,C`	And immediate: $t = $s & C
`or $d,$s,$t`	Or: $d = $s | $t
`ori $t,$s,C`	Or immediate: $t = $s | C
`xor $d,$s,$t`	Exclusive or: $d = $s ^ $t
`xori $t,$s,C`	Exclusive or immediate: $t = $s ^ C
`nor $d,$s,$t`	Nor: $d = ~ ($s | $t)
`slt $d,$s,$t`	Set on less than: $d = ($s < $t)
`sltu $d,$s,$t`	Set on less than unsigned: $d = ($s < $t)
`slti $t,$s,C`	Set on less than immediate: $t = ($s < C)

Bitwise shift

`sll $d,$t,shamt`	Shift left logical immediate: $d = $t << shamt
`srl $d,$t,shamt`	Shift right logical immediate: $d = $t >> shamt
`sra $d,$t,shamt`	Shift right arithmetic immediate
`sllv $d,$t,$s`	Shift left logical: $d = $t << $s
`srlv $d,$t,$s`	Shift right logical: $d = $t >> $s
`srav $d,$t,$s`	Shift right arithmetic

Conditional branch

`beq $s,$t,C`	Branch on equal: if ($s == $t) go to PC+4+4*C
`bne $s,$t,C`	Branch on not equal: if ($s != $t) go to PC+4+4*C

Unconditional jump

`j C`	Jump: PC = PC+4[31:28] . C*4
`jr $s`	Jump register: goto address $s
`jal C`	Jump and link: $31 = PC + 4; PC = PC+4[31:28] . C*4